

**Algorithms, Data Structures & Complexity**  
***Lab 4: Working with Graphs & Dynamic***  
***Programming***

Due on first session of the project for your group

*Federico Pecora, Uwe Köckemann*

**Uwe Köckemann**

### General Hints

- Some of these exercises require to identify how to represent the described problem as a graph and to identify the correct algorithm to solve the problem.
- All required algorithms have been covered in the lecture.

### Handing In

This lab should be completed and shown during the first session of the project for your group. The TA will pass by your seat and evaluate each exercise. Upon successful completion of the lab, for each lab exercise, please provide a text file named `ex_n.txt` with the following content:

- indicate which file(s) implement the algorithm and/or data structure in the exercise;
- a brief explanation of the tests that were carried out to test the implementation;
- instructions on how to execute a test to verify the implemented code;
- answers to any theoretical questions asked in the exercise.

Please submit all lab material collected into an archive (zip, rar, or tar.gz) via a Blackboard message to Uwe Köckemann and Federico Pecora.

**Note:** labs should be done in pairs. Larger groups are *not* allowed. All incidents of plagiarism will be reported. Please write your names on all material you hand in.

### Exercise 1 — Graph Data Structure

Implement a graph data structure using either a matrix representation or an adjacency list representation. Make sure your data structure supports both directed and undirected graphs (you will need both for exercises 2 and 3). Implement the following functions for your graph:

- `createGraph(n)`: Create and return a graph containing  $n$  vertices (each vertex is an integer between 1 and  $n$ ).
- `getNumVertices(G)`: Returns the number of vertices in graph  $G$
- `getNumEdges(G)`: Returns the total number of edges of graph  $G$
- `getNeighbors(G,v)`: Returns all vertices connected to node  $v$  with any edge
- `getInNeighbors(G,v)`: Returns a list of all vertices  $v'$  connected to node  $v$  with an edge  $(v', v)$
- `getOutNeighbors(G,v)`: Returns a list of all vertices  $v'$  connected to node  $v$  with an edge  $(v, v')$
- `addDirectedEdge(v1,v2)`: Create a directed edge between two vertices.
- `addUndirectedEdge(v1,v2)`: Create an undirected edge between two vertices.
- `hasEdge(v1,v2)`: Returns true (1) if there is an edge between  $v1$  and  $v2$  and false (0) otherwise.

Consider using your linked list implementation from Labs 1 & 2 wherever you are asked to return a list.

Create a couple of tests to show that your graph works as intended.

**Exercise 2 — Connectivity**

Implement an algorithm that computes the strongly connected components in a graph  $G = (V, E)$ . Recall that a strongly connected component is a sub-graph in which for each pair of vertices  $(u, v) \in V \times V$  there is a path from  $u$  to  $v$  and vice-versa.

**Exercise 3 — Single-Source Shortest Paths**

Consider a graph  $G = (V, E)$  with weights  $w : E \mapsto \mathbb{R}$ . Implement an algorithm that computes the length of a shortest path from a given vertex  $s \in V$  to all other vertices in  $V$ .

Demonstrate the algorithm on two input graphs of your choice.

Does the algorithm make any assumptions on the input graph?

**Exercise 4 — Testing**

Once you have completed the lab, test an exercise of a colleague and report which tests you conducted and the results of these tests.